Examples 5a, 5b: Synchronized Scanning

Example 5a

Example 5a illustrates the usage of Synchronized Mode. In the code below, the loop reading out the data will be ended when a certain number (10) of profiles are read.

> **Best Practice**
>
> In practice, using the code in Example 5a is not recommended, because a tight loop like this would block the rest of your program. It is included here only for illustration purposes. If, for some reason, the scanner can't measure any points, your program will be stuck in an endless loop. In Example 5b we will look at a better solution, using threads.

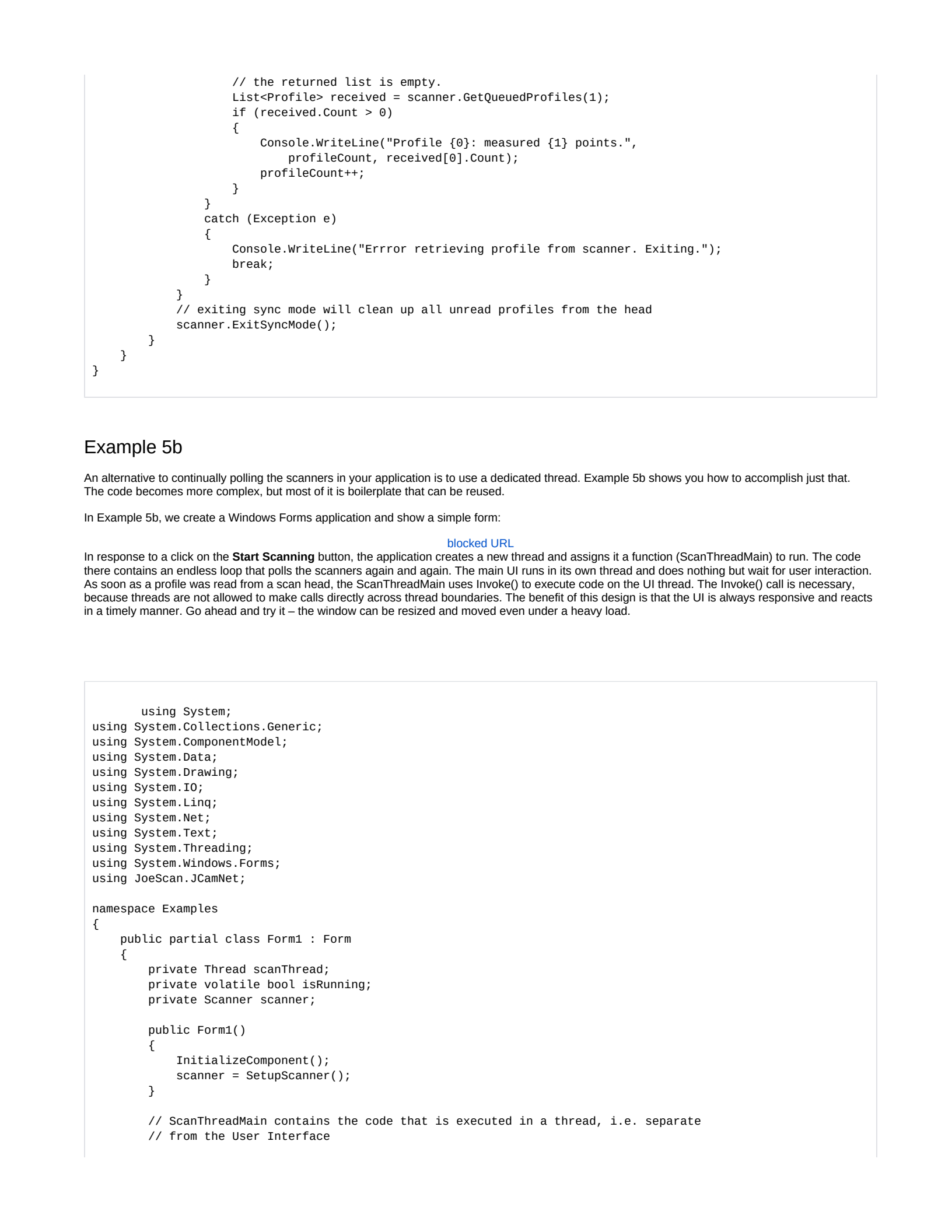
```
      using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading;
using JoeScan.JCamNet;

namespace Examples
{
    class Program
    {
        static void Main(string[] args)
        {
            IPAddress baseIpAddress = new IPAddress(new byte[] { 192, 168, 1, 150 });
            Scanner scanner = null;
            try
            {
                scanner = Scanner.Connect(baseIpAddress, 0);
            }
            catch (Exception e)
            {
                Console.WriteLine("Failed to connect to scanner: {0}, Reason: {1}",
                    baseIpAddress.ToString(), e.Message);
                return;
            }
            TextReader parametersReader;
            using (parametersReader = File.OpenText("param.dat"))
            {
                string parametersString = parametersReader.ReadToEnd();
                try
                {
                    scanner.SetParameters(parametersString, true);
                }
                catch (Exception e)
                {
                    Console.WriteLine("Scanner {0}:{1} did not accept configuration parameters!",
                        scanner.IPAddress.ToString(), scanner.CableID);
                    return;
                }
            }
            // Now we're ready to scan. In this example, we scan until we have received ten Profiles.
            // Be aware that this loop will not end until the ten profiles have been measured, so if you
            // try this code on a real scan head that for some reason can't measure any points
            // (blocked view, wrong window settings etc.), the program will appear to hang!

            scanner.EnterEncoderSyncMode();
            int profileCount = 0;
            while (profileCount < 10)
            {
                try
                {
                    // In Sync Mode, you must use GetQueuedProfiles() instead of GetProfile()
                    // Here we request one profile at a time. If the encoder hasn't triggered a scan yet,
```

```
                    // the returned list is empty.
                    List<Profile> received = scanner.GetQueuedProfiles(1);
                    if (received.Count > 0)
                    {
                        Console.WriteLine("Profile {0}: measured {1} points.",
                            profileCount, received[0].Count);
                        profileCount++;
                    }
                }
                catch (Exception e)
                {
                    Console.WriteLine("Errror retrieving profile from scanner. Exiting.");
                    break;
                }
            }
            // exiting sync mode will clean up all unread profiles from the head
            scanner.ExitSyncMode();
        }
    }
}
```

Example 5b

An alternative to continually polling the scanners in your application is to use a dedicated thread. Example 5b shows you how to accomplish just that. The code becomes more complex, but most of it is boilerplate that can be reused.

In Example 5b, we create a Windows Forms application and show a simple form:



In response to a click on the **Start Scanning** button, the application creates a new thread and assigns it a function (ScanThreadMain) to run. The code there contains an endless loop that polls the scanners again and again. The main UI runs in its own thread and does nothing but wait for user interaction. As soon as a profile was read from a scan head, the ScanThreadMain uses Invoke() to execute code on the UI thread. The Invoke() call is necessary, because threads are not allowed to make calls directly across thread boundaries. The benefit of this design is that the UI is always responsive and reacts in a timely manner. Go ahead and try it – the window can be resized and moved even under a heavy load.

```
       using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading;
using System.Windows.Forms;
using JoeScan.JCamNet;

namespace Examples
{
    public partial class Form1 : Form
    {
        private Thread scanThread;
        private volatile bool isRunning;
        private Scanner scanner;

        public Form1()
        {
            InitializeComponent();
            scanner = SetupScanner();
        }

        // ScanThreadMain contains the code that is executed in a thread, i.e. separate
        // from the User Interface
```

```
private void ScanThreadMain()
{
    // we will use encoder sync mode here
    scanner.EnterEncoderSyncMode();
    // the code on a thread can not directly make calls on the
    // UI thread, hence the need for Invoke()
    this.Invoke(new Action(() => this.textBox.Clear()));
    this.Invoke(new Action(() => this.textBox.AppendText("Scanning Started.\n")));

    // this is an endless loop until the isRunning variable is changed
    // to false from the outside of this thread
    while (isRunning)
    {
        // GetQueuedProfiles() is a shortcut for BeginGetQueuedProfiles()/EndGetQueuedProfiles()
        // See the Example on Hi-Speed Scanning for more information
        List<Profile> received = scanner.GetQueuedProfiles(1);
        if (received.Count > 0)
        {
            this.Invoke(new Action(() =>
                this.textBox.AppendText(String.Format("Profile with {0} points measured.\n",
                received[0].Count))));
        }
    }

    scanner.ExitSyncMode();
    this.Invoke(new Action(() => this.textBox.AppendText("Scanning Stopped.\n")));
}

private void StartButton_Click(object sender, EventArgs e)
{
    if (scanThread == null || !scanThread.IsAlive)
    {
        // we create a new thread here and assign the ScanThreadMain as the code path
        // to be executed when the thread is started
        scanThread = new Thread(new ThreadStart(this.ScanThreadMain))
        {
            // this construct is an initializer list in C#
            IsBackground = true,
            Name = "ScanThreadMain",
            Priority = ThreadPriority.Highest
        };
        isRunning = true;
        scanThread.Start();
    }
}

private void StopButton_Click(object sender, EventArgs e)
{
    StopScanning();
}

private void StopScanning()
{
    isRunning = false;
    if (scanThread != null)
    {
        // wait a moment for the thread to close down
        scanThread.Join(100);
        scanThread = null;
    }
}

private void QuitButton_Click(object sender, EventArgs e)
{
    StopScanning();
    Close();
}

private Scanner SetupScanner()
{
    // for clarity, no error checking here
```

```
            Scanner s = Scanner.Connect(new IPAddress(new byte[] {192, 168, 1, 150}));
            using (TextReader parametersReader = File.OpenText("param.dat"))
            {
                string parametersString = parametersReader.ReadToEnd();
                s.SetParameters(parametersString, true);
            }
            return s;
        }
    }
}
```